**Введение:**

Зачастую при разработке программного обеспечения члены команды сталкиваются со многими проблемами, которые необходимо решать для развития проекта.

**Декомпозиция процесса выполнения задачи:**

Как видно на данной диаграмме, выполнение задачи включает в себя определение статуса и непосредственное создание задачи, решение задачи посредством написания кода, тестирование задачи и внедрение законченного решения в продукт. Результатом этих процессов является законченная и работоспособная часть новой или изменённой логики конечного приложения. И на каждом пути, разработчики сталкиваются с обилием ручной работы.

Каждый из вышеперечисленных блоков так же может быть декомпозирован на слои.

**Основные проблемы:**

Можно выделить основные проблемы при разработке:

– Обсуждение плана разработки или отдельно взятой задачи. Оценка трудоёмкости задач:

– Предоставление результатов обсуждений всей команде;

– Описание и объяснение задач;

– Контроль задач и сотрудников на проекте;

– Информационная защита всех данных на проекте;

– Контроль доступа к данным.

**Необходимые механизмы:**

Данные проблемы можно решить следующим образом:

– Необходим механизм, позволяющий общаться между собой в реальном времени. Результаты общения должны быть видны всем членам команды;

– Необходим механизм оценки трудоёмкости задач;

– Необходим механизм визуализации связей между задачами;

– Необходим механизм полного контроля состояния задач;

– Необходим механизм, для стандартизации всех входных данных;

– Необходим механизм, позволяющий ознакомится с каждым членом команды.

– Необходима возможность для связи со всеми участниками проекта;

– Необходим механизм авторизации и аутентификации.

Для решение всех вышеперечисленных проблем при разработке серверной части приложения необходимо создать следующие инструменты. Будет внедрён механизм авторизации и аутентификации посредством JWT-токенов и стандартных средств языков программирования. Будет разработана база данных для хранения всех программных сущностей и связей между ними. Будет разработан алгоритм создания графа, описывающего связи между задачами, а также алгоритм определения критического пути на графе с учётом оценок трудоёмкости задач. Будет разработана API для предоставления клиентской части всех необходимых инструментов, таких как доступ к базе данных, взаимодействия с сущностями в ней. Будет разработан механизм разграничения на роли пользователей. Будет внедрён механизм кэширования часто используемых данных.

**Граф состояний проекта**:

При одновременной работе нескольких программистов задачи могут выполняться параллельно. В таком случае разные разработчики выполняют разные задачи. Однако они в силу различных причин тратят на выполнение задач неодинаковое количество времени.

Например, рассмотрим следующий граф. Состояния 1 и 2 являются соответственно начальным и конечным состояниями некоторого разрабатываемого приложения в некотором отрезке времени. Для перехода из состояния 1 в состояние 2 необходимо выполнить все задачи, обозначенные буквами. В процессе параллельной разработки задачи A1, B1-2, C1-2 могут быть выполнены существенно быстрее задач A2, C3. В таком случае разработчики, закончившие раньше, начинают простаивать, что является неэффективным использованием времени.

Разрабатываемое приложение устраняет причины лишних затрат времени.

Для начала введём понятие критического пути. Критический путь состоит из самой длительной последовательности задач от начала проекта до его завершения. Тогда время, которое необходимо затратить на разработку всего проекта (*T*), равно времени, затрачиваемом на каждую задачу, лежащую на критическом пути.

**Работа с API:**

Я занимался разработкой API, с которым будет способен взаимодействовать абсолютно любой клиент. В данном случае предоставлен пример работы сервера с задачами. Это пример запроса через Postman. За счёт того, что серверная часть реализована как API, в целом, приложению не важно, кто запрашивает данные – браузер, другой API или ещё какой-нибудь источник, что обеспечивает несомненную гибкость.

**Преимущества:**

1. Легка в освоении, не в ущерб функциональным особенностям.
2. Быстрота отклика по сравнению с самыми популярными аналогами Azure и Jira.
3. Дружелюбный интерфейс согласно некоторым требованиям “средств взаимодействия человека и компьютера”.
4. Легковесность, при использовании локального варианта, в то время как вышеперечисленные конкуренты не имеют локальной версии.
5. Широкий ассортимент предоставляемых функций со стороны сервера (в Azure приходится делать несколько последовательных запросов для получения некоторых данных).
6. Документация внутри функций, у Microsoft, к сожалению, встречается документация в виде имён функций “function1-function2”, что не несёт никакой информационной нагрузки.
7. Открытый исходный код, который можно модифицировать под свои нужды.

Что изменилось за год:

**Так как мы знали, что университет хотел бы передать проект следующих студентам, мы делали упор и на то, чтобы сделать их работу наиболее комфортной.**

1. Переработка проекта под React-Redux-Saga. Это сделано для предоставления возможности текущим и будущим разработчикам без труда расширять приложение.

2. Настроен CI/CD, что облегчит работу текущих и следующих разработчиков в несколько раз.

3. Переработка архитектуры под трёхуровневую на серверной части и под сервисную архитектуру для Redux-Saga.

**Улучшения для разработчиков**  
1. Добавлена возможность создавать больше одного проекта, таким образом между членами команды и проектами становится связь многие ко многим. До этого введения, для работы над несколькими проектами нужно было создавать несколько учётных записей.

2. Добавлены диалоговые окна, предотвращающих ошибочные действия пользователей.

3. Добавилась возможность при необходимости перейти на режим работы со спринтами (по методологии Agile).

4. Глобальная оптимизация, сокращающая время ответа сервера на 34% (использовали автоматизированные тесты и прибегли к помощи коллег для замеров времени).

5. Приложение теперь можно развернуть на облачном хостинге Azure, что технически означает глобальное внедрение. То есть теперь любой человек может через интернет использовать данный продукт.

6. Были найдены и устранены критические ошибки.

7. Переход на современный набор библиотек (MaterialUI, Dapper).

8. Усовершенствована система общения путём добавления уведомлений, приходящих на почту члену команды; заведён специальный личный электронный ящик для приложения.

9. Работа над логотипом приложения.

10. Переработка базы данных: нормализация, добавление и изменение таблиц.

11. Кэширование данных на сервере.